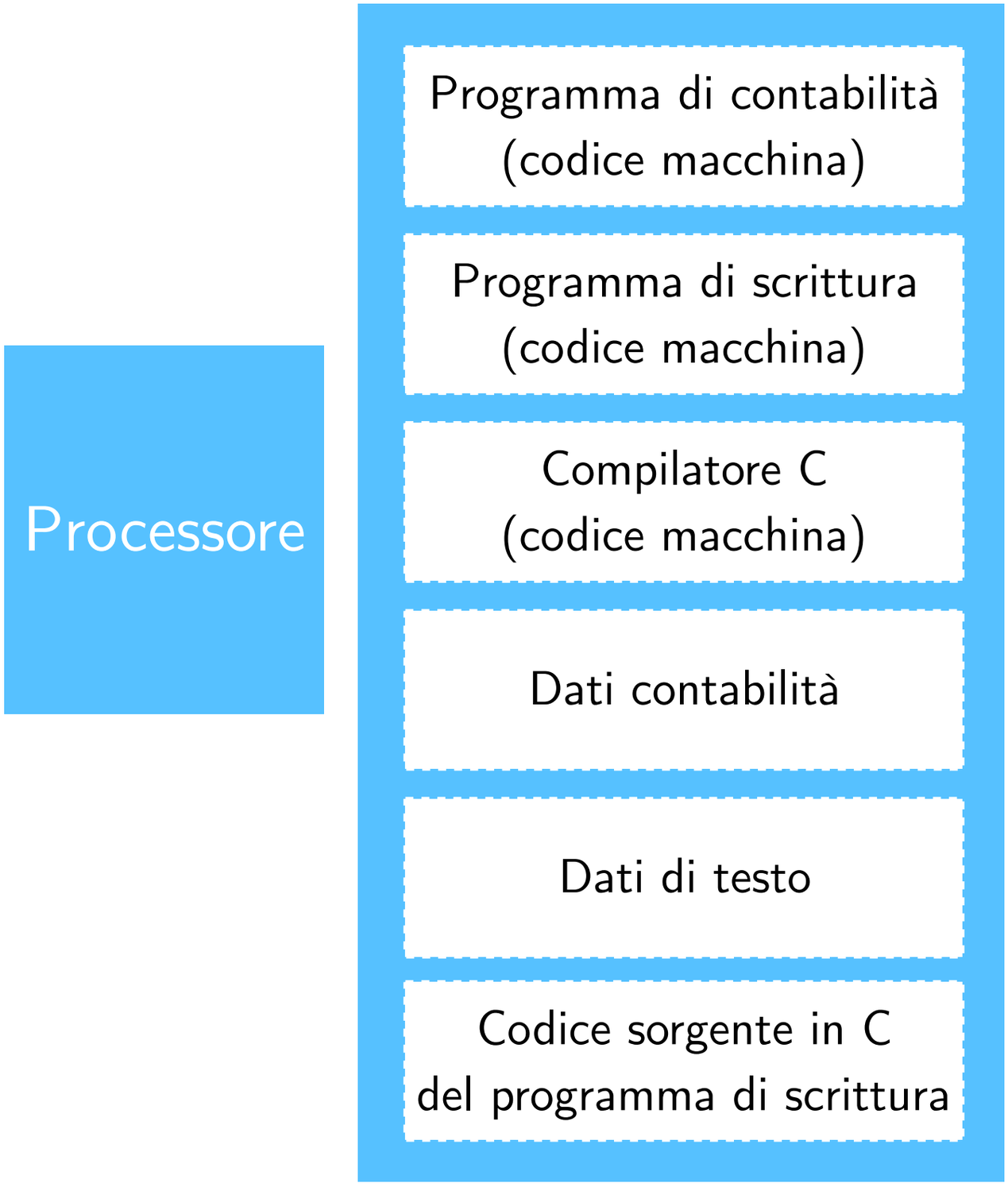
Stored Program Computer



- Le istruzioni sono rappresentate in binario, proprio come i dati
- I dati e le istruzioni sono immagazzinati in memoria
- I programmi posso operare su programmi
  - Compilatori, linker, ...
- La compatibilità binaria permette a programmi compilati di funzionare su computer diversi
  - ISA standardizzate

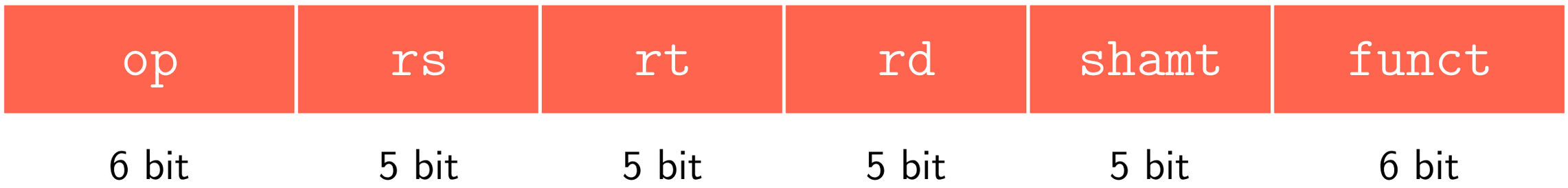
Operazioni logiche

- Istruzioni per la manipolazione bit a bit

Operazione	C	Java	MIPS
Shift a sinistra	<<	<<	`sll`
Shift a destra	>>	>>>	`srl`
AND bit a bit	&	&	`and, andi`
OR bit a bit	|	|	`or, ori`
NOT bit a bit	~	~	`nor`

- Utili per estrarre e inserire gruppi di bit in una word

Operazioni di shift



- `shamt` (shift amount): di quante posizioni scorrere
- Scorrimento logico a sinistra
  - scorrimento a sinistra riempiendo con bit a 0
  - `sll` (shift left logical) di *i* bit moltiplica per 2^i
- Scorrimento logico a destra
  - scorrimento a destra riempiendo con bit a 0
  - `srl` (shift right logical) di *i* bit divide per 2^i (solo unsigned)

Operazione AND

- Utile per mascherare bit in una word
  - Seleziona alcuni bit e pone gli altri a 0

```
and $t0, $t1, $t2
```

```
$t2  0000 0000 0000 0000 0000 0000 1101 1100 0000
$t1  0000 0000 0000 0000 0000 0011 1100 0000 0000
$t0  0000 0000 0000 0000 0000 0000 1100 0000 0000
```

Operazione OR

- Utile per includere bit in una word
  - Pone alcuni bit a 1 e lascia gli altri inalterati

```
or $t0, $t1, $t2
```

```
$t2  0000 0000 0000 0000 0000 0000 1101 1100 0000
$t1  0000 0000 0000 0000 0000 0011 1100 0000 0000
$t0  0000 0000 0000 0000 0000 0011 1101 1100 0000
```

Operazione NOT

- Utile per complementare bit a bit in una parola
  - Cambia 0 in 1 e 1 in 0
- Il MIPS ha l'istruzione NOR a 3 operandi
  - `a NOR b == NOT (a OR b)`

```
nor $t0, $t1, $zero
```

```
$t1  0000 0000 0000 0000 0011 1100 0000 0000
$t0  1111 1111 1111 1111 1100 0011 1111 1111
```

Operazioni condizionali

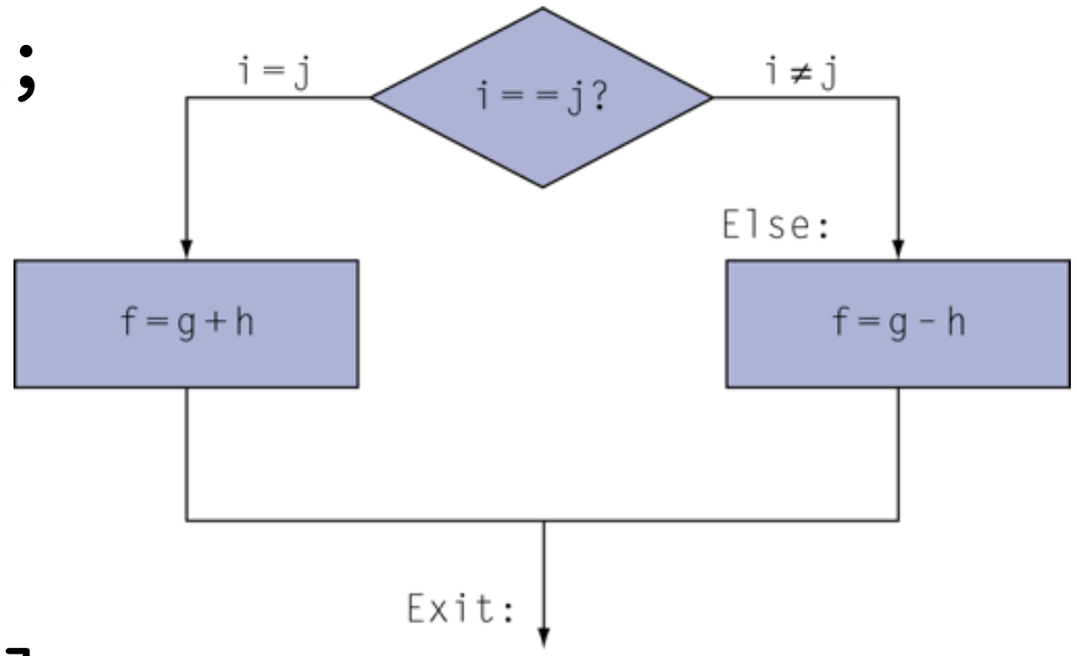
- Salto (branch) a un'istruzione etichettata se la condizione è vera
  - Altrimenti continuare in sequenza
- `beq rs, rt, L1`
  - se (`rs == rt`) salta all'istruzione con etichetta `L1`
- `bne rs, rt, L1`
  - se (`rs != rt`) salta all'istruzione con etichetta `L1`
- `j L1`
  - salta incondizionatamente all'istruzione con etichetta `L1`

Compilare comandi condizionali

- Codice C:

```
if (i == j) f = g + h;
else f = g - h;
```

  - f, g, ... in $s0, $s1, ...



- Codice MIPS compilato:

```
        bne $s3, $s4, Else
        add $s0, $s1, $s2
        j Exit
Else:   sub $s0, $s1, $s2
Exit:   ...
```

L'assemblatore calcola gli indirizzi

Compilare comandi di ciclo

- Codice C:

```
while (save[i] == k)
    i += 1;
```

  - `i` in `$s3`, `k` in `$s5`, indirizzo di `save` in `$s6`

- Codice MIPS compilato:

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j Loop
Exit: ...
```

Altre operazioni condizionali

- Porre il risultato a 1 se la condizione è vera
  - Altrimenti porlo a 0
- `slt rd, rs, rt`
  - se (`rs` $<$ `rt`) `rd` $=$ 1; altrimenti `rd` $=$ 0
- `slti rd, rs, constant`
  - se (`rs` $<$ `constant`) `rt` $=$ 1; altrimenti `rt` $=$ 0
- Usato in combinazione con `bne` e `beq`

```
slt $t0, $s1, $s2  # if ($s1 < $s2)
bne $t0, $zero, L  # salta a L
```

Progettazione delle istruzioni di salto

- Perchè non `blt`, `bge`, etc.?
- L'hardware per $<$, $\geq$, ... è più lento di quello per $=$, $\neq$
  - Combinato con un salto richiede più lavoro per istruzione, e quindi un clock più lento
  - Tutte le istruzioni sono penalizzate!
- `beq` e `bne` sono i casi più comuni
- Questo è un buon compromesso di progetto

Invocazione di procedure

- Passi necessari:
  1. Disporre i parametri nei registri
  2. Trasferire il controllo alla procedura
  3. Acquisire spazio per la procedura
  4. Eseguire le operazioni della procedura
  5. Disporre il risultato nel registro per l'invocante
  6. Ritornare al punto di invocazione

Uso dei registri

- `$a0 - Sa3`: argomenti (registri 4 – 7)
- `$v0, $v1`: valori di ritorno (registri 2 e 3)
- `$t0 - $t9`: valori temporanei
  - Possono essere sovrascritti dalla procedura chiamata
- `$s0 - $s7`: valori salvati
  - Devono essere salvati/rispristinati dalla procedura chiamata
- `$gp`: *global pointer* per i dati statici (registro 28)
- `$sp`: *stack pointer* (registro 29)
- `$fp`: *frame pointer* (registro 30)
- `$ra`: indirizzo di ritorno (registro 31)

Istruzioni per invocare una procedura

- Invocazione di una procedura: *jump and link*

  `jal EtichettaProcedura`

  - L'indirizzo dell'istruzione successiva è posto in `$ra`
  - Salto all'indirizzo di destinazione
- Ritorno da una procedura: *jump register*

  `jr $ra`

  - Copia `$ra` nel program counter
  - Può anche essere usato per salti calcolati
    - per esempio, nei comandi case/switch

Esempio con procedura foglia

- Codice C:

```
int esempio_foglia(int g, int h, int i, int j)
{
    int f;
    f = (g + h) - (i + j);
    return f;
}
```

- Argomenti `g`, ..., `j` copiati in `$a0`, ... `$a3`
- Variable locale `f` in `$s0` (quindi dobbiamo salvare `$s0` sullo stack)
- Risultato copiato in `$v0`

Esempio con procedura foglia

- Codice MIPS:

```
esempio_foglia:
```

```
    addi $sp, $sp, -4
    sw   $s0, 0($sp)
```

Salvare $s0 sullo stack

```
    add  $t0, $a0, $a1
    add  $t1, $a2, $a3
    sub  $s0, $t0, $t1
```

Corpo della procedura

```
    add  $v0, $s0, $zero
```

Risultato

```
    lw   $s0, 0($sp)
    addi $sp, $sp, 4
```

Ripristinare $s0

```
    ja $ra
```

Ritorno

Esempio con procedura non-foglia

- Procedure che invocano altre procedure
- Per invocazioni annidate, la procedura chiamante deve salvare sullo stack:
  - Il suo indirizzo di ritorno
  - Qualsiasi argomento e variabile temporanea necessaria al termine dell'invocazione
- Ripristinare tutto il necessario dallo stack al termine dell'invocazione

Esempio con procedura non-foglia

- Codice C:

```
int fattoriale(int n)
{
    if (n < 1)
        return 1;
    else
        return n * fattoriale(n - 1);
}
```

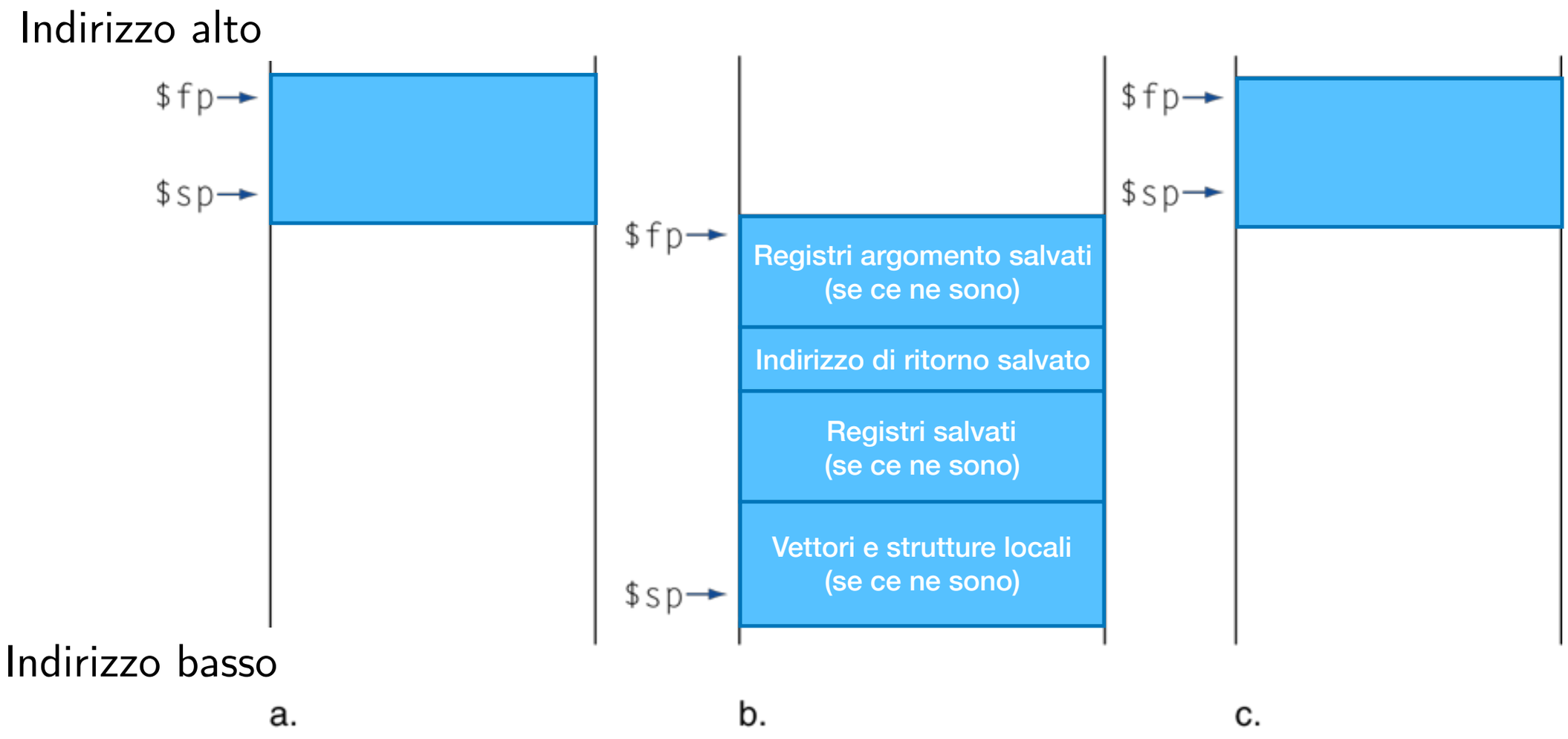
- Argomento `n` copiato in `$a0`
- Risultato copiato in `$v0`

Esempio con procedura non-foglia

- Codice MIPS:

```
Facts:
    addi $sp, $sp, -8       # prerarare lo stack per 2 elementi
    sw   $ra, 4($sp)        # salvare l'indirizzo di ritorno
    sw   $a0, 0($sp)        # salvare l'argomento
    slti $t0, $a0, 1        # test per n < 1
    beq  $t0, $zero, L1
    addi $v0, $zero, 1      # se è vero, il risultato è 1
    addi $sp, $sp, 8        #   estrarre 2 elementi dallo stack
    jr   $ra                #   e ritornare
L1: addi $a0, $a0, -1       # altrimenti decrementare n
    jal  fact               #   e invocare ricorsivamente
    lw   $a0, 0($sp)        # ripristinare n originale
    lw   $ra, 4($sp)        #   e l'indirizzo di ritorno
    addi $sp, $sp, 8        # estrarre 2 elementi dallo stack
    mul  $v0, $a0, $v0      # moltiplicare per il risultato
    jr   $ra                # e ritornare
```

Dati locali sullo stack



- Dati locali allocati dal chiamato
  - per esempio, variabili C automatiche
- Frame della procedura (record di attivazione)
  - Usato da alcuni compilatori per gestire lo spazio dello stack